Rabinizer 3: Safraless Translation of LTL to Small Deterministic Automata*

Zuzana Komárková[1] and Jan Křetínský[2]

[1] Faculty of Informatics, Masaryk University, Brno, Czech Republic
[2] IST, Austria

Abstract. We present a tool for translating LTL formulae into deterministic ω-automata. It is the first tool that covers the whole LTL that does not use Safra's determinization or any of its variants. This leads to smaller automata. There are several outputs of the tool: firstly, deterministic Rabin automata, which are the standard input for probabilistic model checking, e.g. for the probabilistic model-checker `PRISM`; secondly, deterministic *generalized* Rabin automata, which can also be used for probabilistic model checking and are sometimes by orders of magnitude smaller. We also link our tool to `PRISM` and show that this leads to a significant speed-up of probabilistic LTL model checking, especially with the generalized Rabin automata.

1 Introduction

The automata-theoretic approach to model checking is a very successful concept, which found its way to real industrial practice. The key idea is that a property to be checked on a given system is transformed into an automaton and the product of the automaton and the original system is then examined. Since real systems are often huge it is important that the automata used are very small, so that the product is not too large to be processed or even fit in the memory. Therefore, a lot of effort has been invested in transforming popular specification languages, such as *linear temporal logic* (LTL) [Pnu77], to small automata [Cou99, DGV99, EH00, SB00, GO01, GL02, Fri03, BKŘS12, DL13].

The property automata are usually non-deterministic Büchi automata (NBA) as they can express all LTL properties and are quite succinct. However, for purposes of quantitative probabilistic LTL model checking or of LTL synthesis, *deterministic* automata are needed [BK08]. To this end, we can transform NBA to deterministic Rabin automata (DRA) using Safra's determinization procedure or its variants [Saf88, Pit06, Sch09] implemented in [Kle, KNP11, TTH13]. The disadvantage of this approach is that the blow-up is often exponential even for

* This research was funded in part by the European Research Council (ERC) under grant agreement 267989 (QUAREM), the Austrian Science Fund (FWF) project S11402-N23 (RiSE), and the Czech Science Foundation, grant No. P202/12/G061. Jan Křetínský is on leave from Faculty of Informatics, Masaryk University, Brno, Czech Republic.

F. Cassez and J.-F. Raskin (Eds.): ATVA 2014, LNCS 8837, pp. 235–241, 2014.

simple formulae despite various heuristics [KB07]. Therefore, practically more efficient procedures have been designed for *fragments* of LTL [KE12, KLG13, BBKS13]. A comparison of currently available translators into deterministic automata [Kle, GKE12, KLG13, BBKS13] can be found in [BKS13].

While our technique for the $(\mathbf{F}, \mathbf{G})$-fragment [KE12, GKE12] was extended to some larger fragments [KLG13], an occurrence of the $\mathbf{U}$ operator in the scope of the $\mathbf{G}$ operator posed a fundamental problem for the approach. Recently [EK14], we have shown how to modify the techniques of [KE12] to the whole LTL, using a more complex procedure. In this paper, we present its implementation together with several optimizations: `Rabinizer 3`, the first tool to translate LTL formulae directly to deterministic automata not employing any automata determinization procedure. Thus after partial solutions of `Rabinizer` [GKE12] based on [KE12], and `Rabinizer 2` [KLG13], we finally reach our ultimate goal set up from the very start.

Firstly, we optimize the construction of the state space given in [EK14]. We also implement the construction of the acceptance condition of the automata. As the condition is defined by an exponentially large description, several optimizations were needed to compute it efficiently and to obtain an equivalent small condition. Our optimizations lead to automata and acceptance conditions no larger than those generated by tools for fragments of LTL.

Furthermore, we provide an interface between our tool and `PRISM` [KNP11], the leading probabilistic model checker, resulting in a faster tool for probabilistic LTL model checking. While `PRISM` uses DRA produced by a re-implementation of `ltl2dstar` [Kle], our tool produces not only DRA, but also *generalized DRA* (DGRA). They are often much smaller and can also be used for probabilistic model checking for virtually no extra cost [CGK13]. Moreover, since the algorithm for probabilistic LTL model checking through DGRA has the same structure as for DRA, it was possible to implement it reusing the `PRISM` code.

`Rabinizer 3` as well as the extended `PRISM` are available at [R3].

2 Tool and Experimental Results

Principles and Optimizations. The key idea of the approach of [EK14] is to have (i) one "master" automaton monitoring the formula that at each step needs to be satisfied, and (ii) one "slave" automaton for each subformula of the form $\mathbf{G}\psi$ monitoring whether ψ holds true at *all but finitely many* positions in the word.[1] They all run synchronously in parallel; the slaves are organized recursively, providing information to the slaves of larger formulae and finally also to the master.

Example 1. Consider $\varphi = (a \wedge \mathbf{FG}b) \vee (\mathbf{F}a \wedge \mathbf{FG}c)$. Upon reading $\{a, b\}$, the master moves to $\mathbf{FG}b \vee \mathbf{FG}c$ and the slave for $\mathbf{FG}b$ records it has seen a b. If we

[1] This approach bears some similarity with temporal testers [KPoR98, PZ08], which non-deterministically guess satisfaction at each point and later check the guesses. In contrast, Mojmir automata [EK14] used here are deterministic and thus can provide the information to the master only through acceptance.

read $\emptyset$ instead, the master would move to $\mathbf{F}a \wedge \mathbf{FG}c$. Apparently, in this state, it makes no sense any more to monitor whether $\mathbf{FG}b$ holds or not. Moreover, we can postpone checking $\mathbf{FG}c$ until we see an a.

Both observations of the previous example lead to optimizations saving unnecessary states. The former was considered already in [KLG13]. The latter is similar to [BBDL$^+$13], where a similar effect is achieved by graph analysis of the automata. In contrast, here we can detect the same situation much more easily using the logical structure of the state space. Thus for instance, for the formula $\mathbf{F}a \wedge \mathbf{GF}(b \wedge \mathbf{XXX}b)$ the size drops from 16 states in [EK14] to 9 here. Further, an optimization of the initial states of slaves leads to a similar saving, e.g., for $\mathbf{GF}((a \wedge \mathbf{XXX}a) \vee (\neg a \wedge \mathbf{XXX}\neg a))$ from 15 to 8 states.

A major advantage of our approach is that it can generate deterministic *generalized* Rabin automata [KE12, CGK13].

Definition 1 (DGRA). *A* deterministic generalized Rabin automaton (DGRA) *is a deterministic ω-automaton with an acceptance condition of the form*

$$\bigvee_{i=1}^{k} (Fin_i, \bigwedge_{j=1}^{k_i} Inf_i^j)$$

A run visiting exactly set S of states infinitely often is accepting, if for some i, $S \cap Fin_i = \emptyset$ and $S \cap Inf_i^j \neq \emptyset$ for all $j = 1, \ldots, k_i$.

Hence, DRA are DGRA with all k_i equal to 1. Similarly, we can define *transition-based* DGRA (DTGRA), where the acceptance sets Fin_i, Inf_i^j are sets of transitions. We use both variants and the transition-based acceptance often saves even more, see Table 1 for examples of fairness constraints. The lower part of the table illustrates the effect of the optimizations: the size of DTGRA due to unoptimized [EK14] for ψ_1 and ψ_2 is 11 and 32, respectively, compared to 3 and 16 using the new optimizations. For more experiments, see the web-page of the tool [R3].

The generalized Rabin acceptance condition arises naturally from the product of Rabin conditions for each slave and one global co-Büchi condition. Unfortunately, due to the global consition it is a disjunction over all *subsets* of **G**-subformulae and various subsets of slaves' states. Therefore, it is large. However, after simplifying the pairs, removing pairs simulated by other pairs and several other steps, we often decrease the number of pairs down to the actual Rabin index [KPB95], i.e. the minimal possible number for the given language, as illustrated in Table 1.

Outputs. Given a formula, Rabinizer 3 can output the corresponding DTGRA, DGRA and DRA. Several output formats are available, such as the `ltl2dstar` format, the `dot` format for visualization, or the `PRISM` format. Optional labels on states display the internal logical structure of the automaton. Transitions can be displayed either explicitly or more compactly using BDDs, e.g. $a + b$ stands for three transitions, namely under $\{a\}$, $\{b\}$, and $\{a, b\}$.

Table 1. Experimental comparisons on fairness constraints (upper part) and two formulae of [EK14] (lower part). We display number of states and acceptance pairs for `ltl2dstar` and `Rabinizer 3` producing different types of automata, all with the same number of pairs. Here $\psi_1 = \mathbf{FG}(((a \wedge \mathbf{XX}b) \wedge \mathbf{GF}b)\mathbf{UG}(\mathbf{XX}!c \vee \mathbf{XX}(a \wedge b)))$ and $\psi_2 = \mathbf{G}(!q \vee (((!s \vee r) \vee \mathbf{X}(\mathbf{G}(!t \vee r) \vee !r\mathbf{U}(r \wedge (!t \vee r))))\mathbf{U}(r \vee p) \vee \mathbf{G}((!s \vee \mathbf{XG}!t))))$, the latter being φ_{40} "1 cause-2 effect precedence chain" of Spec Patterns [SP].

Formula	`ltl2dstar`		`Rabinizer 3`			
	DRA states	pairs	DRA st.	DGRA st.	DTGRA st.	pairs
$\mathbf{FG}a \vee \mathbf{GF}b$	4	2	4	4	1	2
$(\mathbf{FG}a \vee \mathbf{GF}b) \wedge (\mathbf{FG}c \vee \mathbf{GF}d)$	11324	8	21	16	1	4
$\bigwedge_{i=1}^{3}(\mathbf{GF}a_i \to \mathbf{GF}b_i)$	1 304 706	10	511	64	1	8
$\bigwedge_{i=1}^{3}(\mathbf{GF}a_i \to \mathbf{GF}a_{i+1})$	153 558	8	58	17	1	8
ψ_1	40	4	4	4	3	1
ψ_2	314	7	21	21	16	4

Probabilistic Model Checking. We follow up on our experimental implementation of [CGK13] for DGRA and DRA. We provide Java classes allowing for linking any tool with an appropriate output text format to be used in `PRISM`.

Since we also produce transition-based DGRA, our experimental results reveal an interesting observation. Although state-based DGRA are larger than their transition-based counterpart DTGRA, the respective product is not much larger (often not at all), see Table 2. For instance, consider the case when the only extra information that DGRA carries in states, compared to DTGRA, is the labeling of the last transition taken. Then this information is absorbed in the product, as the system's states carry their labeling anyway. Therefore, in this relatively frequent case for simpler formulae (like the one in Table 2), there is no difference in sizes of products with DGRA and DTGRA.

Table 2. Model checking Pnueli-Zuck mutex protocol with 5 processes (altogether 308 800 states) from the benchmark set [KNP11] for the property that either all processes 1-4 enter the critical section infinitely often, or process 5 asks to enter it only finitely often

	`ltl2dstar` DRA	`R.3` DRA	`R.3` DGRA	`R.3` DTGRA
Automaton size (and nr. of pairs)	196 (5)	11 (2)	33 (2)	1 (2)
Product size	13 826 588	1 100 608	308 800	308 800

Further, notice that the DGRA in Table 2 is larger than the DRA obtained by degeneralization of DTGRA and subsequent transformation to a state-based automaton. However, the product with the DGRA is of the size of the original system, while for DRA it is larger! This demonstrates the superiority of generalized Rabin automata over standard Rabin automata with respect to the product size and thus also computation time, which is superlinear in the size. For details, further experiments, and the implementation, see [R3].

3 Conclusion

We present the first tool translating the whole LTL directly to deterministic ω-automata, while not employing any automata determinization procedure. This often results in much smaller DRA. Moreover, the power of DGRA is now available for the whole LTL as well. Together with our modification of `PRISM`, this allows for further speed up of probabilistic model checking as demonstrated by experimental results.

Acknowledgement. We would like to thank Javier Esparza, Vojtěch Forejt, Mojmír Křetínský, Marta Kwiatkowska, and Dave Parker for discussions and feedback and Andreas Gaiser and Ruslán Ledesma Garza for pieces of code we could reuse.

References

[BBDL+13] Babiak, T., Badie, T., Duret-Lutz, A., Křetínský, M., Strejček, J.: Compositional approach to suspension and other improvements to LTL translation. In: Bartocci, E., Ramakrishnan, C.R. (eds.) SPIN 2013. LNCS, vol. 7976, pp. 81–98. Springer, Heidelberg (2013)

[BBKS13] Babiak, T., Blahoudek, F., Křetínský, M., Strejček, J.: Effective translation of LTL to deterministic Rabin automata: Beyond the (F, G)-fragment. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 24–39. Springer, Heidelberg (2013)

[BK08] Baier, C., Katoen, J.-P.: Principles of model checking. MIT Press (2008)

[BKŘS12] Babiak, T., Křetínský, M., Řehák, V., Strejček, J.: LTL to Büchi automata translation: Fast and more deterministic. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 95–109. Springer, Heidelberg (2012)

[BKS13] Blahoudek, F., Křetínský, M., Strejček, J.: Comparison of LTL to deterministic Rabin automata translators. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR-19 2013. LNCS, vol. 8312, pp. 164–172. Springer, Heidelberg (2013)

[CGK13] Chatterjee, K., Gaiser, A., Křetínský, J.: Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 559–575. Springer, Heidelberg (2013)

[Cou99] Couvreur, J.-M.: On-the-fly verification of linear temporal logic. In: Wing, J.M., Woodcock, J. (eds.) FM 1999. LNCS, vol. 1708, pp. 253–271. Springer, Heidelberg (1999)

[DGV99] Daniele, M., Giunchiglia, F., Vardi, M.Y.: Improved automata generation for linear temporal logic. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 249–260. Springer, Heidelberg (1999)

[DL13] Duret-Lutz, A.: Manipulating LTL formulas using spot 1.0. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 442–445. Springer, Heidelberg (2013)

[EH00] Etessami, K., Holzmann, G.J.: Optimizing Büchi automata. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 153–167. Springer, Heidelberg (2000)

[EK14] Esparza, J., Křetínský, J.: From LTL to Deterministic Automata: A Safraless Compositional Approach. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 192–208. Springer, Heidelberg (2014)

[Fri03] Fritz, C.: Constructing Büchi automata from linear temporal logic using simulation relations for alternating Büchi automata. In: Ibarra, O.H., Dang, Z. (eds.) CIAA 2003. LNCS, vol. 2759, pp. 35–48. Springer, Heidelberg (2003)

[GKE12] Gaiser, A., Křetínský, J., Esparza, J.: Rabinizer: Small deterministic automata for LTL(F,G). In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 72–76. Springer, Heidelberg (2012)

[GL02] Giannakopoulou, D., Lerda, F.: From states to transitions: Improving translation of LTL formulae to Büchi automata. In: Peled, D.A., Vardi, M.Y. (eds.) FORTE 2002. LNCS, vol. 2529, pp. 308–326. Springer, Heidelberg (2002)

[GO01] Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 53–65. Springer, Heidelberg (2001), at `http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/`

[KB07] Klein, J., Baier, C.: On-the-fly stuttering in the construction of deterministic ω-automata. In: Holub, J., Žďárek, J. (eds.) CIAA 2007. LNCS, vol. 4783, pp. 51–61. Springer, Heidelberg (2007)

[KE12] Křetínský, J., Esparza, J.: Deterministic automata for the (F,G)-fragment of LTL. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 7–22. Springer, Heidelberg (2012)

[Kle] Klein, J.: ltl2dstar - LTL to deterministic Streett and Rabin automata, `http://www.ltl2dstar.de/`

[KLG13] Křetínský, J., Garza, R.L.: Rabinizer 2: Small deterministic automata for LTL\GU. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 446–450. Springer, Heidelberg (2013)

[KNP11] Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)

[KPB95] Krishnan, S.C., Puri, A., Brayton, R.K.: Structural complexity of ω-automata. In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 143–156. Springer, Heidelberg (1995)

[KPoR98] Kesten, Y., Pnueli, A., Raviv, L.-O.: Algorithmic verification of linear temporal logic specifications. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 1–16. Springer, Heidelberg (1998)

[Pit06] Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: LICS, pp. 255–264 (2006)

[Pnu77] Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57 (1977)

[PZ08] Pnueli, A., Zaks, A.: On the merits of temporal testers. In: Grumberg, O., Veith, H. (eds.) 25MC Festschrift. LNCS, vol. 5000, pp. 172–195. Springer, Heidelberg (2008)

[R3] Rabinizer 3, https://www7.in.tum.de/~kretinsk/rabinizer3.html

[Saf88] Safra, S.: On the complexity of ω-automata. In: FOCS, pp. 319–327 (1988)

[SB00] Somenzi, F., Bloem, R.: Efficient Büchi automata from LTL formulae. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 248–263. Springer, Heidelberg (2000)

[Sch09] Schewe, S.: Tighter bounds for the determinisation of Büchi automata. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 167–181. Springer, Heidelberg (2009)

[SP] Spec Patterns: Property pattern mappings for LTL, http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml

[TTH13] Tsai, M.-H., Tsay, Y.-K., Hwang, Y.-S.: GOAL for games, omega-automata, and logics. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 883–889. Springer, Heidelberg (2013)